

UNITED STATES PATENT APPLICATION

**SWITCH MANAGEMENT SYSTEM AND METHOD**

**INVENTORS**

**Abraham Rabindranath Matthews**

**Anna Berenberg**

Schwegman, Lundberg, Woessner, & Kluth, P.A.

1600 TCF Tower

121 South Eighth Street

Minneapolis, Minnesota 55402

ATTORNEY DOCKET 1384.006US1

Attorney Docket 1384.006US1

006US1.091384.006US1

## SWITCH MANAGEMENT SYSTEM AND METHOD

### Field of the Invention

The present invention is related to networking systems, and more particularly to a system and method for managing a switch within a wide area network (WAN).

### Background Information

Internet or WAN service providers are operating in a crowded marketplace where cost effectiveness is critical. Operational costs present a significant challenge to service providers. Cumbersome, manual provisioning processes are the primary culprits. Customer orders must be manually entered and processed through numerous antiquated back-end systems that have been pieced together. Once the order has been processed, a truck roll is required for onsite installation and configuration of Customer Premises Equipment (CPE), as well as subsequent troubleshooting tasks.

Presently, the delivery of firewall services requires the deployment of a specialized piece of Customer Premises Equipment (CPE) to every network to be protected. This model of service delivery creates an expensive up-front capital investment, as well as significant operational expenses that are associated with onsite installation and management of thousands of distributed devices. The results are service delivery delays, increased customer start-up costs and/or thinner service provider margins.

The slow and expensive process of deploying firewall services cuts into margins and forces significant up-front charges to be imposed on the customer. In order to be successful in today's market, service providers must leverage the public network to offer high-value, differentiated services that maximize margins while controlling capital and operational costs. These services must be rapidly provisioned and centrally managed so that time-to-market and, more importantly, time-to-revenue are minimized. Traditional methods of data network service creation, deployment, and management present significant challenges to accomplishing these goals, calling for a new network

service model to be implemented.

Enterprise customers are increasingly demanding cost-effective, outsourced connectivity and security services, such as Virtual Private Networks (VPNs) and managed firewall services. Enterprise networks are no longer segregated from the outside world; IT managers are facing mounting pressure to connect disparate business units, satellite sites, business partners, and suppliers to their corporate network, and then to the Internet. This raises a multitude of security concerns that are often beyond the core competencies of enterprise IT departments. To compound the problem, skilled IT talent is an extremely scarce resource. Service providers, with expert staff and world-class technology and facilities, are well positioned to deliver these services to enterprise customers.

What is needed is a system and method for providing managed network services that are customizable for each customer's need. Furthermore, what is needed is a system and method for controlling such managed network services.

### **Summary of the Invention**

According to one aspect of the present invention, a system and method of managing a switch includes installing a switch having a plurality of processor elements, installing an operating system on each processor element, creating a system virtual router and configuring the processor elements from the system virtual router.

According to another aspect of the present invention, a switch management system includes an object manager and a distributed management layer, wherein the object manager communicates with objects through the distributed management layer.

### **Description of the Preferred Embodiments**

In the following detailed description of the preferred embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.



While service providers recognize the tremendous revenue potential of managed firewall services, the cost of deploying, managing and maintaining such services via traditional CPE-based methods is somewhat daunting. Service providers are now seeking new service delivery mechanisms that minimize capital and operational costs while enabling high-margin, value-added public network services that are easily provisioned, managed, and repeated. Rolling out a network-based managed firewall service is a promising means by which to accomplish this. Deploying an IP Service Delivery Platform in the service provider network brings the intelligence of a managed firewall service out of the customer premises and into the service provider's realm of control.

One such IP Service Delivery Platform 10 is shown in Fig. 1. In the embodiment shown in Fig. 1, IP Service Delivery Platform 10 includes three distinct components: an intelligent, highly scalable IP Service Processing Switch 12, a comprehensive Service Management System (SMS) 14 and a powerful Customer Network Management (CNM) system 16. Service Management System (SMS) 14 is used to enable rapid service provisioning and centralized system management. Customer Network Management (CNM) system 16 provides enterprise customers with detailed network and service performance systems, enable self-provisioning. At the same time, system 16 eases IT managers fears of losing control of managed network services.

In one embodiment, such as is shown in Fig. 2 for a network-based managed firewall service model, the service provider replaces the high-capacity access concentration router at the POP with an IP Service Processing Switch 12. This is a higher-capacity, more robust, and more intelligent access switch, with scalable processing up to 100+ RISC CPUs. Just as with the access router, additional customer access capacity is added via installing additional port access blades to the IP Service Processing Switch chassis. Unlike conventional access routers, however, additional processor blades can be added to switch 12 to ensure wire-speed performance and service processing.

The intelligence resident in IP Service Processing Switch 12 eliminates the need

to deploy CPE devices at each protected customer site. Deployment, configuration, and management of the managed firewall service all take place between IP Service Processing Switch 12 and its Service Management System 14. In the embodiment shown in Fig. 2, Service Management System 14 resides on a high-end UNIX platform at the service provider NOC.

In one embodiment, the customer has the ability to initiate service provisioning and augmentation via a web-based Customer Network Management tool residing, e.g., at the customer's headquarters site. This is an entirely different service delivery paradigm, requiring little or no truck rolls and little or no on-site intervention.

In one embodiment, switch 12 is a 26-slot services processing switch that marries scalable switching, routing and computing resources with an open software architecture to deliver computationally-intense IP services such as VPNs with scalable high performance. In one embodiment, switch 12 has a high-speed 22 Gbps redundant dual counter-rotating ring midplane. Slots are configured with four types of Service Blades: Control, Access, Trunk and Processor blades with specialized processing which enables a range of high-performance services including route forwarding, encryption and firewalls.

Service providers can use switch 12's virtual routing capabilities, and its ability to turn IP services into discrete and customized objects, to segment and layer services for the first time for tens of thousands of discrete subscriber corporations. In addition, processor capacity can be added to switch 12 by adding new processor blades.

In one embodiment switch 12 includes an operating system which dynamically distributes services to switch 12 processors.

In one embodiment, the 26-slot services processing switch corrects for failures using the redundant counter-rotating ring midplane.

In one embodiment, each Service Blade automatically fails-over to a backup.

One embodiment of a switch 12 is described in "System and Apparatus for Delivering Security Services," filed herewith, the description is incorporated herein by reference.

In one embodiment, switch 12 is designed to integrate seamlessly into a SP's



In one embodiment, the IPSX object database consists of two types of databases: Global (managed on Master Control Blade by OMORIG) and distributed local databases (managed by OMORI agents on every PE present in the system). In one such embodiment, the global database is a superset of the extracts from local databases.

Objects represent a basic unit of management for purposes of fault tolerance, computational load balancing, etc. One or more adjacent protocol modules can be placed into a single object. It is also possible that a module is split across two objects.

In IP, each host has a globally unique IP Address. Additionally each type of transport on top of IP has a globally unique Protocol ID. Each application on top of a transport has a Local Port Number that is unique locally. Thus an application instance in the network is uniquely identified by the tuple

<IP Address, Protocol ID, Port Number>

In switch 12, each Processing Element (PE) has a globally unique PEID. Each type of object in a PE has a globally unique Type ID. Within each type, objects are assigned locally unique numbers or ID. Thus within a switch 12, each object is uniquely identified (analogous to IP applications) by

<PEID, Object Type, Local Object ID>

The format of the Local Object ID is dependent on the object type. Mostly, driver and IO Layer objects have IDs that are constructed based on physical attributes. The physical attributes used are

blade	A globally unique Blade ID (e.g. slot number).
port	A locally unique Port ID for all ports within a blade.
channel	A locally unique Channel ID for all channels within a port. (This may be dynamically variable as in Channellized DS3.)
vcid	A locally unique ID that is assigned by the Link Layer protocol agent, e.g., is a FR DLCI.

The following function is used to initialize an Object ID for any object created by the Object Manager. (e.g. object id type is OBJ\_ID\_TYPE\_OBJECT).

```
void obj_id_init (
    object_id_t          *id,
    object_type_t        type,
    object_group_id_t    group,
    local_object_id_t    object);
```



The following function is used to initialize an Object ID for any object created by the IO Layer. IO Layer objects are created either at system startup time by the device driver subsystem or by a Link Layer protocol module in response to an IOCTL. The important thing to note is that it is not created by Object Manager 24.

```
void obj_id_init_physical (
    object_id_t      *id,
    object_type_t     type,
    physical_id_t     blade,
    physical_id_t     port,
    physical_id_t     channel,
    physical_id_t     vcid);
```

Group is an aggregation point for all objects that comprises the VR. Group and VR have one-to-one mapping. A Group encompasses objects, which are located in different address spaces. Group Id, which identifies a group, is unique in the scope of a single switch 12.

Figure 5 shows the distinction between an Object Class and an Object Group. Both are collections of objects. As shown in Figure 5, an object class is a set of objects that have the same type signature and behavior (e.g. Applications Class 50, TCP/IP Class 52 and Interfaces Class 54). In contrast, for an object group, the constituent objects do not necessarily have the same type signature and behavior (e.g. Object Groups 1 to 3). There can be multiple objects of the same class in an object group (e.g. Object Group 2 has two objects of Interface Class). On the other hand, an object group need not have an object of each class (e.g. Object Group 3 does not have an object of Interface Class).

In one embodiment, OMCD 40 is the agent, which interfaces to the Configuration Manager. As shown on Figure 6 OMCD 40 manages 1) the Global list of VPN in system 10; and 2) the list of VRs per VPN. The caveats for VPN and VRs are:

- VPN ID equal to 0 is illegal;
- Global uniqueness of VPN ID across IPSX systems is the responsibility of the Service Management System (SMS).

In one embodiment, OMCD 40 creates a vpn descriptor every time Configuration managers request VPN creation. Every VPN is identified by a unique VPN ID. In one embodiment, each Virtual Router (VR) is identified by a VR ID, which is the IP Address of the VR. VR ID is unique in the VPN context. When Configuration Manager requests creation of an existing VR in the VPN, VR creation request is rejected. Otherwise a vr descriptor will be created.

There are several types of the VR:

- 1) ISP (Internet Service Provider) VR: Typically there is 1 such VR for a single switch 10.
- 2) Control VR: There can be only one Control VR for a single switch 10. This VR is used to host the management applications such as SNMP, Telnet etc.
- 3) Customer VR: There are several Customer VRs in a single switch 10. Typically, there is one Customer VR per customer service point.

Detailed VR creation process is described below.

OMORIG agent 42 runs on every Control Blade, whether it is Master or Standby Blade. OMORI local sends the change only to Master. Control Blade Redundancy feature, described below **Error! Reference source not found.** takes care of replicating and synchronizing OMORIG database from Master to Standby.

OMORIG 42 provides several mappings of Object Ids. It manages lists of object Ids, which are located on the same address space, lists of object Ids which belong to the same group, a sorted Global object ID list and an unsorted Global object ID list. The OID link is shown on the Figure 7. The OID link data structure and OMORIG API are described in Appendix A.

OMORI is the OM agent. OMORI runs on every processing node and manages local objects and forwards IOCTLS to another object, whether local or remote. OMORI for each object creates `object_descriptor_t`, which has all the management information.

As shown on Figure 8, OMORI manages a global list of local object descriptors 60, a global list of groups 62, and a list of object descriptors per group 64.

Each change in the OMORI database is propagated to the OMORIG, which runs on the Active Master. OMORI sends separate messages, varying by message tag, per each action to be taken to change Global Database.

OMORI additionally serves the request from the object on information about any other object. If requested object local to OMORI then it finds all the data in the local database otherwise OMORI agent forwards such a request to OMORIG, which has validated data.

The creation and deletion of objects in an object group needs to be coordinated. The issues to be dealt with are as follows. First, an object may need to IOCTL another object for correct setup or shutdown. We need to ensure that all default objects for the group are present.

Second, an object when using a cached pointer must be sure that it has not become stale.

Every OMORI maintains a state machine for each local object. Each object is supposed to take an appropriate action on every state change notification from the OMORI. Object State Transition Diagram is in the Figure 9 and detailed description is on the Table 1.

The caveats for the object states are as follows. First, in *init* state, the object's base data structure is created and initialized. Essential auxiliary data structures may also be created and initialized.

Third, in *active* state, non-essential auxiliary data structures and system resources are activated. The object may cache pointers. The object can initiate and respond to IOCTLs.

STATE	EVENT	ACTION
INIT	Receive Create Object Request	Cal constructor for the object class. Transit to STOPPED
STOPPED	RECV ACTIVATE request	Send ACTIVATE_OBJECT generic IOCTL, if completed with SUCCESS transit to ACTIVE.
STOPPED	RECV DESTROY request	Send DESTROY_OBJECT generic IOCTL, transit to DEAD, remove object descriptor from list, free it.
ACTIVE	RECV DESTROY request	Transit to STOPPED state. Send DESTROY_OBJECT generic IOCTL, transit to DEAD, remove object descriptor from list, free it.
ACTIVE	RECV ACTIVATE request	
STOPPED	RECV ADD to Group /REMOVE from Group request	Modifies group membership as requested
ACTIVE	RECV ADD to Group /REMOVE from Group request	Transit to STOPPED state. Modifies group membership as requested

### Table 1. Object State Machine

Distributed Messaging Layer (DML) 22 is used to provide inter-processor communication and isolated channels for data and control messages as is shown in Figure 10.

OMORIG and OMORI communicate via predefined DML channel DML\_CHAN\_DATA. All IPNOS nodes in the system are members of DML\_CHAN\_DATA. During initialization process OMORI register to DML receive function, which will be called every time a new packet arrives on DML\_CHAN\_DATA. OMORIG and OMORI are DML applications and therefore they are notified on every dynamic event in the system.

There are four types of dynamic events indicated by DML. These are:

- Peer Up – new IPNOS node detected and reachable.
- Peer Down – existing IPNOS node became unreachable
- Master Up – new Master elected in the system
- Master Down – existing Master became unreachable

On peer down event OMORI agent aborts all the pending transactions associated with the peer that went down. In addition, on a peer down event OMORIG destroys in its database all the objects that are local to the peer that went down. After that a scrub of the database is done. This includes destroying all groups which do not have any objects in them and destroying any VR associated with that group.

On peer up event and master up event OMORIG agent runs global database update protocol shown in Figure 11. In addition, on peer up event OMORIG agent initiates a database update for local objects of the new peer. OMORIG maintains state machine per OMORI. Global Database Update State Transition Diagram is shown in Figure 11. A detailed description of the transitions is in Table 2.

STATE	EVENT	ACTION
START	TIMEOUT && (request count < MAX)	Send update request
START	TIMEOUT && (request count > MAX)	Peer did not reply. Update FAILED Transit to FINISH state.
START	RCV UPDATE GROUP message	Transit to UPDATE GROUP state. Set last update equal to the current time.
START	RCV UPDATE OBJECT message	Transit to UPDATE OBJECT state. Set last update equal to the current time.
UPDATE GROUP	RCV UPDATE GROUP message	Set last update equal to the current time. Modify Database
UPDATE GROUP	RCV UPDATE GROUP DONE message	Transit to UPDATE OBJECT state. Set last update equal to the current time.
UPDATE GROUP	TIMEOUT && (delay > MAX)	Transit to FINISH state.
UPDATE OBJECT	TIMEOUT && (delay > MAX)	Transit to FINISH state.

UPDATE OBJECT	RECV UPDATE OBJECT DONE message	Transit to FINISH state.
UPDATE OBJECT	RECV UPDATE OBJECT message	Set last update equal to the current time. Modify Database

**Table 2. Global Database Update Protocol**

The same protocol is initiated on a master up event by the OMORIG agent to all peers that are known at that moment.

As described above, when peer goes down all virtual routers (VRs) as well as groups and objects, associated with that peer, are removed from the Global Database. If for some reason switch 12 becomes partitioned and then coalesces back, the problem with dangling objects arises, because all the objects still exists on the isolated peer. To address this a database reconciliation process is provided.

On a peer up event, the global database update protocol is started. When an update group message is received and the group is not found in the group list then

- 1) Look up VR by VPN ID and VR ID from the update message. If not found, then need to recreate dependencies by the following algorithm:
  - Check whether VPN with VPN ID, from the update message, exists. If not then create VPN with the specified VPN ID.
  - Check whether VR with VR ID, from the update message, exists. If not then create VR with the specified VR ID.
  - Create group with ID received from the update message.
- 2) VPN/VR found: Send message to the OMORI, which send an update message to remove, specified group.

Transaction Layer 46 in Figure 10 is used to provide management of request/ reply transaction and has *exactly once* semantics, with guaranteed termination. There is a global list of requests on each processor. Each request is identified by unique index. When request is being sent a callback function could be provided if reply is expected. One embodiment of Transaction Layer Data structures and an associated API are described in Appendix A.

Object creation and communication will be described next.

In Figure 12, the communication for object creation is shown. IP object with OID 1 requests Firewall object to be created. OM creates object descriptor and based on the specified class of the object (e.g. Firewall), OM finds the appropriate *constructor* function in the object class table and constructs the new object. The same algorithm is used for management communications between objects (IOCTL). Based on the class id of the destination object

appropriate *control* function from the object class table is called. It is the responsibility of the object implementers is to supply handlers for all supported IOCTLs.

The IOCTL mechanism is described below. Typically IOCTL between objects is used for Management Plane communications. For Data Plane communications between objects, object to object channels are used.

Objects can be created in three different ways:

- **REGISTERED:** Created as system comes up (e.g. drivers) and registered to the OMORI with object id, having physical location meaning.
- **CREATED BY OM:** Created by Object Manager. In this case OMORI creates a locally unique (in the scope of this address space) object ID that in conjunction with address space id gives unique object id inside of the system. To create an object, the constructor function based on the object class will be called.
- **ASSIGNED and REGISTERED:** This is hybrid of two cases described above. Object created without OM, but requests OM to assign unique Object ID to it and registers with this ID to OMORI. Typically this is used by the Resource Manager.

OM 24 issues control codes when invoking the control method for an object. To ease the management of these control codes, a module based code scheme is used.

Every code is composed of an implicit module identifier and a module specific code identifier. The macro *OBJ\_CTL\_CODE(module, module\_specific\_code)* is used to construct the object control code.

```
#define OBJ_CTL_CODE(m,c)    (((m) << 24) | (c))
#define OBJ_CTL_MODULE(c)    ((c) >> 24)
```

Generic IOCTL are primarily, used by Object Manager 24, to inform all the objects in the specified group of some state change. For example, when user requests to delete VR, before removing all the objects in the VR's group, STOP\_OBJECT generic IOCTL is sent to every object in the group. MODULE\_ALL is used as a module identifier for all the generic IOCTLs.

Every object should support the following generic IOCTLs:

```
ACTIVATE_OBJECT
STOP_OBJECT
DESTROY_OBJECT
```

**096847**

**DEREGISTERED:** Objects which were registered on creation will be deregistered on destruction

**DESTROYED:** There is no explicit destructor to destroy an object, instead generic IOCTL DESTROY\_OBJECT is send to object, which is to be destroyed.

The IOCTL mechanism provides a reliable transaction oriented inter-object communication that is suitable for management and control traffic. However, the IOCTL based communication is not fast or efficient. For protocol data traffic, a lighter, faster and efficient mechanism is needed.

In one embodiment, object channels provide a *Point-to-point* (P-P) communication mechanism that is based on a *send-and-forget* model of programming. Packets arriving at an object channel are delivered to the object asynchronously.

Objects maintain a list of *Connection End Points* (CEP). Each CEP is assigned an index that is unique within the scope of the object. Since the object's OID is globally unique, a globally unique CEP-ID is generated by the tuple

 $\langle OID, Type, index \rangle$ 

The *type* parameter is used to distinguish between different classes of CEPs. (E.g. the IP Forwarding object has CEPs for Virtual Interfaces and CEPs to cryptographic resources.) The CEP is represented in IPNOS by the *obj comm t* data structure.

Each object allocates a CEP (which is typically embedded within an object specific structure). The object then initializes the CEP by the function

```
extern int      obj_init_channel (
    obj_comm_t      *channel,      /* Channel to init */
    void             *object, /* Object channel is associated with */
    obj_comm_service_f *service    /* Rx Packet service handler */
);
```

The *service* parameter for *obj\_init\_channel* is an upcall handler function that is called when a packet is received on the channel. The *object* parameter is passed as the first parameter of the upcall handler, and is typically used to identify the data structure that the channel is embedded in.

After a CEP has been initialized, it can be connected to another CEP via

```
int    obj_associate_channel (
        obj_comm_t          *local_chan,    /* Local channel */
        obj_cep_id_t        *local_cep,     /* Local channel ID */
        obj_cep_id_t        *remote_cep     /* Remote channel ID */
    );
```

A CEP that is connected can be disconnected via

```
int obj_disassociate_channel (
        obj_comm_t          *local_chan     /* Local channel */
    );
```

Sometimes it is necessary that a CEP be loopbacked to itself. This can be done by

```
int obj_loopback_channel (
        obj_comm_t          *local_chan,    /* Local channel */
        obj_cep_id_t        *local_cep     /* Local channel ID */
    );
```

Figure 13 shows the linkages when two CEPs within the same address space are connected. Note that

1. The CEP has a *send* and a *receive* section (of type *obj\_comm\_params\_t*).
2. The *send* section of Obj-1 is updated with the contents of the *receive* section of Obj-2.
3. The *send* section of Obj-2 is updated with the contents of the *receive* section of Obj-1.
4. The *Remote CEP-ID* of each object (of type *obj\_cep\_id\_t*) contains the CEP-ID of the CEP at the other end.
5. The field *remote\_srv\_cb* points to the remote CEP. (This is not used when the CEPs are in different address spaces.

When Obj-1 sends a packet to Obj-2, it becomes a function call. The overhead is very little. The sequence of actions that takes place when *obj\_associate\_channel* is called is shown in Table 3.

Step	Local CEP Object	IPNOS	Remote CEP Object
1	<i>obj_associate_channel</i> ( <i>local_chan</i> ,		



	local_cep_id, remote_cep_id)		
2		omori_obj_ioctl_by_id (&remote->object, remote_cep_id- >object.group, OBJ_CTL_CODE (remote_cep_id - >module_id, GET_CEP_ADDR), &cep, sizeof (get_cep_addr_t) Remote_chan = cep.address	
3			In GET_CEP_ADDR IOCTL handler, return CEP's address.
4		Copy CEP IDs to remote and local	
5		Setup local channel pointers	
6		Setup remote channel pointers	

**Table 3: Connecting local CEPs**

Figures 14 and 15 show how services can be *pushed* onto a channel that has been established. (Note that the services and parameters should be *pushed* after a connection has been established. Services and service parameters from an earlier connection can not be assumed to be in effect.)

To enable a service, use the function

```
int      obj_update_service_params_on_channel (
    obj_comm_t          *channel,
    int                  service_id,
    int                  direction,
    int                  operation,
    void                 *params
);
```

To disable a service, use the function

```

int    obj_disable_service_on_channel (
        obj_comm_t                *channel,
        int                        service_id,
        int                        direction
    );

```

To update the parameters for a service, use the function

```

int    obj_update_service_params_on_channel (
        obj_comm_t                *channel,
        int                        service_id,
        int                        direction,
        int                        operation,
        void                      *params
    );

```

Note that in Figure 14, only the local CEP is modified when a service is enabled on transmit. In Figure 15 on the other hand, the remote CEP is modified when a service is enabled on receive.

The services that are currently supported are:

OBJ_COMM_SRV_NONE	This is never used. It is used to indicate the CEP base.
OBJ_COMM_SRV_UNSPECIFIED	This is never used. May be used to indicate errors.
OBJ_COMM_SRV_REMOTE	This service provides transport between PEs (aka address spaces). This service is automatically pushed, by IPNOS, on both receive and transmit at both ends of the channel, when the CEPs are in different PEs.
OBJ_COMM_SRV_LOCALQ	In a VI-VI connection, this is used to breakup the function call chain. It is used only when the VI CEPs are both in a single PE.
OBJ_COMM_SRV_RAQ	This is used to enforce a specific rate. The integration interval used is that of a single OS Clock Tick. It is currently not used. In the future a better algorithm based on the <i>Leaky Bucket</i> should be used.

**SECRET**

5/3/1

26/5/20

In one embodiment, OMORI cache table is designed to use a *closed hashing* algorithm (also known as *open addressing*). In a closed hashing system, if collision occurs, alternate cells

are tried until the empty cell is found. In one embodiment, closed hashing with linear probing is used. In one such embodiment, limited search is added such that, in case of collision only a limited number of cells will be tried. If empty cell is not found, then a new entry will replace the collided one.

In one embodiment, all elements in the OMORIG as well as in the OMORI database are managed using double linked circular list. As the number of elements in the list increases rises, however, the problem of search latency becomes an issue. In one embodiment, therefore, lists (which supposedly have large number of elements) are modified to the hash table. Open hashing is used for this purpose. Open hashing is to keep a list of all elements that hash to the same value.

One embodiment of a Control Blade Redundancy algorithm will be discussed. As noted above, in one embodiment, system 10 is designed to provide Fault Tolerance. In one such embodiment, each Control Blade runs management modules such as Command Line Interface (CLI) and Simple Network Management Protocol (SNMP), which allows configuration of system 10. Each of these modules retrieves data from the OM Global Database that resides on the Control Blade. Global database is constructed from the distributed OM Local Databases, which are stored on every processing node in the system.

In one embodiment, each switch 12 has at least two Control Blades. In the event of Control Blade failure, system management and configuration are done using the backup Control Blade. Thus NOS 20 provides a Control Blade Redundancy (CBR) service. This document discusses the protocol used to provide a synchronized backup of the OM Global Database as part of the CBR service.

In the following description,

**Master** – Processing Engine 0 (PE 0) on the Control Blade (CB) which is being use to manage and configure the system and participates in the message passing communication.

**Slave** – Processing Engine 1-3 (PE 1-3) on the CB and PE0-4 on Access or Processor Blades which participates in the message passing communication.

**Standby** - Processing Engine 0 (PE 0) on the Control Blade (CB) which can be used to manage and configure the system in the case of Master failure and participates in the message passing communication.

**Peer** – any Processing Engine which participates in the message passing communication

As noted above, NOS 20 consists of the several layers. Figure 24 shows only layers, which are relevant to the Database Redundancy problem that is discussed in this document.

Control Ring Driver 23 – notifies upper layer on the following events:

- blade up: new blade inserted in the system and became operational
- blade down: blade removed from the system and became non-operational;
- master blade up: new CB inserted and Control Ring decided that this is a Master
- standby blade up: new CB inserted and Control Ring decided that this is a Standby
- slave blade up: new blade inserted and this is not CB.

Distributed Messaging Layer (DML) 22 is message passing model to provide inter connectivity between processing nodes and channel management. DML 22 provides a reliable group communication based on message passing infrastructure by implementing:

- reliable sequenced layer
- supports of channels that provide independent communication universes
- Group operation on the channel like send, recv barrier synchronization and broadcast.
- Dynamic group membership that reflects dynamic state of the system.

Object Manager (OM) 24 is a module, which manages VPN, VR, objects and object groups in the system. Provides an IOCTL like mechanism for reliable fault tolerant messaging between objects that is typically used for management function. This

uses the DML channel "DML\_CHAN\_WORLD". This mechanism was described above.

CB Channel 25 is a DML channel whose members are the dynamic set of Control Blades present and intercommunicating in the system.

Control Blade Redundancy (CBR) 26 is a module, which provides redundant Global Database on the Standby blades; CBR 26 is a DML application that receives notification from DML on all UP/DOWN events.

In one embodiment, Control Blade redundancy (CBR) 26 is designed to create and maintain replicas of the Master Control Blade management information on the Standby Control Blades and to reuse that information in the case of failure of the current Master and election of a new Master. Control Ring Driver 23 normally elects the new Master. If the Control Ring detection mechanism fails, however, a software-based leader election protocol implemented by DML 22 will elect the new Master. This redundancy is illustrated in Figure 25.

An important part of the management information is the OM Global Database. A key issue of the CBR is the consistency of OM Global Database. The OM Global Database is synchronized in two ways: bulk updates and flash updates. Bulk updates are used in CBR 26 on dynamic events like peer up/down. Flash updates are used to propagate individual change in the database (like a VR being created or deleted).

There are four Data Types which CBR protocol supports: Virtual Private Network (VPN), Virtual Router (VR), GROUP (an internal representation of the VR; set of all objects belonging to VR), and Object ID (OID).

CBR protocol provides sequential messaging per Data Type. If Standby receives update message with sequence number, which is not equal to the one expected then Standby sends message about it to Master and Master restarts update from the beginning. Note that DML provides a sequenced reliable transport and this should not happen normally. It could happen if the underlying SRTP Point-to-Point link resets as a result of timeout.

As a DML application CBR 26 is notified of events happening in the system. The events indicated are *peer up*, *peer down*, *master up*, *master down*.

On peer down event CBR does not need to take any action, OM on every Control Blade will update its database.

On master up/master down event CBR also does not need to take any action, because master up event always comes with peer up/peer down event where all the actions were taken.

On peer up event Master will dump its own Global Database to the all Standby Nodes. The dump algorithm is described in the Transition Diagram.

Figure 26 shows Master's State Transition Diagram. Master maintains state of each node participating in the CBR protocol. Master itself does not transition from the READY State. When peer up event occurs for a standby a new CBR node is added to the list and state is initialized to READY. In Figure 26, DT denotes Data Types (described above). ALL\_MARKED is a bitmap that is used to keep track of the MARK replies for the specific Data Type. When all replies arrived bitmap is equal to bitmask, which means that all Data Types were marked.

ALL\_FINISHED is a bitmap that is used to keep track of the FINISH replies for the specific Data Type. When all replies arrived bitmap is equal to bitmask, which means that all Data Types were finished

DUMP Master State Transition Table is given on the Table 4.

STATE	EVENT	ACTION
READY	PEER UP	Send MARK request for each Data Type Transit to START DUMP
READY	RECV DUMP request from Standby (invalid message sequence number)	Send MARK request for each Data Type, clear ALL_MARKED Transit to START DUMPs
vSTART DUMP	RECV MARK reply for one of the Data Types &&! ALL Data Types MARKED	Modify ALL_MARKED to include replied peer
START DUMP	RECV MARK reply for one of the Data Types && ALL Data Types MARKED	Transit to DUMP_IN_PROGRESS State. For each Data Type send

		DUMP DATA; For each Data Type send FINISH DATA; Transit to FINISH_DUMP;
START DUMP	RECV DUMP request from Standby (invalid message sequence number)	Send MARK request for each Data Type, clear ALL MARKED
DUMP_IN_ PROGRESS	RECV DUMP request from Standby (invalid message sequence number)	Send MARK request for each Data Type Transit to START DUMP
FINISH DUMP	RECV FINISH reply for one of the Data Types && !ALL Data Types FINISHED	Modify ALL_FINISHED to include replied peer
FINISH DUMP	RECV FINISH reply for one of the Data Types && ALL Data Types FINISHED	Transit to READY
FINISH DUMP	RECV DUMP request from Standby (invalid message sequence number)	Send MARK request for each Data Type Transit to START DUMP

**Table 4. DUMP Master State Transition Table.**

Standby Node maintains the state transitions shown on Figure 27 and in Table 5 only for itself. When CBR 26 is initialized state is READY

STATE	EVENT	ACTION
READY	RECV MARK request for one of the Data Types	Send MARK reply for this Data Type Transits to START DUMP
READY	RECV invalid message sequence number for flash updates	Send DUMP request to Master
READY	RECV ADD request for one of the Data Types and invalid message sequence number	Send DUMP request to Master
READY	RECV ADD request for one of the Data Types	Add data to the Database
READY	RECV DELETE request for one of the Data Types and invalid message sequence number	Send DUMP request to Master
READY	RECV DELETE request for one of the Data Types	Delete data from the Database



START DUMP	RECV MARK request for one of the Data Types && ALL Data Types MARKED	Transit to DUMP_IN_PROGRESS State. For each Data Type send DUMP DATA; For each Data Type send FINISH DATA; Transit to FINISH DUMP;
START DUMP	RECV invalid message sequence number	Send DUMP request to Master Transit to READY
DUMP_IN_PROGRESS	RECV DUMP request for one of the Data Types	if this is a new item then add to Database, otherwise modify existing one.
DUMP_IN_PROGRESS	RECV invalid message sequence number	Send DUMP request to Master Transit to READY
FINISH DUMP	RECV FINISH request for one of the Data Types && !ALL Data Types FINISHED	Modify ALL_FINISHED to include replied peer Send FINISH reply for this Data Type
FINISH DUMP	RECV FINISH request for one of the Data Types && ALL Data Types FINISHED	Send FINISH reply for this Data Type Transit to READY
FINISH DUMP	RECV invalid message sequence number	Send DUMP request to Master Transit to READY

**Table 5. Dump Standby State Transition Table**

Master sends MARK request for each data type to this peer and transits to the START\_DUMP state. When Standby receives mark request for one of the data types it transits to START\_DUMP state, marks all existing elements of specified type and sends reply back to the Master. In its turn master delays start of dump until it receives MARK replies for all the Data Types. When all the replies are received Master transits to DUMP\_IN\_PROGRESS state and dumps all elements of its Database to the Standby peer. Standby receives DUMP message and updates its data in the Database and unmarks updated element. When DUMP is done Mater sends to Standby FINISH message and transits to the FINISH\_DUMP state. After receiving FINISH message Standby transits to the FINISH\_DUMP state, deletes all the elements in the Database,

00663483 091300  
DOCT60" E34E9960

which are left marked and sends FINISH reply to the Master. Standby stays in this state until finish procedure done for all Data Types and then goes into READY STATE. Master remains in the FINISH state until FINISH replies are received for all Data Types. If Standby receives message with invalid sequence number it sends DUMP\_REQUEST to the master and transits to READY state from the state where Standby was when message arrived. Upon receiving DUMP\_REQUEST Master transits to START\_DUMP state.

OMORIG on Master blade calls CBR 26 to update all known Standby Peers for every change in the Global Database. There are two types of changes: ADD and DELETE. When Standby receives ADD update it looks up in its replicated database for a requested data type by the specified ID. If the specified data item is found then it is modified with received information. If search fails then new data item is created and added to the database. On the DELETE request Standby finds requested data type and removes it without removing semantic dependencies. The OM on Master observes all semantic dependencies when it calls CBR to update a particular Data Type.

Standby Peer maintains simple FSM for flash updates as shown on Figure 27.

Flash Updates as well as Bulk updates are sequential and loss of a message causes restart of the Dump procedure.

526  
A7  
B. A representative CBR API and associated data structures is shown in Appendix

In one embodiment, to be absolutely sure that Standby OM Global Database is a mirror from the Master OM Global Database, periodic updates are used. Standby can run periodic update infrequently. Periodic update is based on the consistency rules checks. If one of the consistencies rules fails, then Standby requests Bulk update from the Master.

Consistency rules for OM Global Database are:

- For every VPN there is a unique ID in the system.
- For every VR there is a unique combination of the VPN ID and VR ID.
- Every VR has a unique ID in the scope of VPN.
- For every group there is a VR to which this group belongs.

- For every object there is a group to which this object belongs.
- Every object has a unique ID in the system.
- Value of counter "Number of VRs in the VPN descriptor" is equal to total number of VRs per VPN.
- Value of the counter "Number of objects in the group descriptor" is equal to the total number of objects in the group.
- Total number of objects across all groups is equal to the total number of objects across address spaces, and it is equal to total number of objects in the system.
- Number of objects in the sorted global list is equal to number of objects in the global hash, and it is equal to total number of objects in the system.
- For every object class in the system there is a corresponding entry in the class table.

In one embodiment, the service provider's security staff consults with the customer in order to understand the corporate network infrastructure and to develop appropriate security policies (Note: this is a similar process to the CPE model). Once this has been accomplished, the NOC security staff remotely accesses the IP Service Processing Switch (using the Service Management System) at the regional POP serving the enterprise customer, and the firewall service is provisioned and configured remotely.

### Conclusion

System 10 as described above enables the service provider to leverage the enterprise's existing services infrastructure (leased lines and Frame Relay PVCs) to deliver new, value-added services without the requirement of a truck roll. All firewall and VPN functionality resides on the IP Service Processing Switch at the POP, thus freeing the service provider from onsite systems integration and configuration and effectively hiding the technology from the enterprise customer. Firewall inspection and access control functions, as well as VPN tunneling and encryption, take place at the IP Service Processing Switch and across the WAN, while the enterprise's secure leased

line or Frame Relay PVC access link remains in place. The customer interface is between its router and the IP Service Processing Switch (acting as an access router), just as it was prior to the rollout of the managed firewall service. Additionally, the customer has visibility into and control over its segment of the network via the CNM that typically resides at the headquarters site.

The network-based firewall model also enables service providers to quickly and cost-effectively roll out managed firewall solutions at all enterprise customer sites. As a result, secure Internet access can be provided to every site, eliminating the performance and complexity issues associated with backhauling Internet traffic across the WAN to and from a centralized secure access point. As the IP Service Delivery Platform is designed to enable value-added public network services, it is a carrier-grade system that is more robust and higher-capacity than traditional access routers, and an order of magnitude more scaleable and manageable than CPE-based systems. The platform's Service Management System enables managed firewall services, as well as a host of other managed network services, to be provisioned, configured, and managed with point-and-click simplicity, minimizing the need for expensive, highly skilled security professionals and significantly cutting service rollout lead-times. The Service Management System is capable of supporting a fleet of IP Service Processing Switches and tens of thousands of enterprise networks, and interfaces to the platform at the POP from the NOC via IP address. Support for incremental additional platforms and customers is added via modular software add-ons. Services can be provisioned via the SMS system's simple point and click menus, as well as requested directly by the customer via the CNM system. Deployment of a robust IP Service Delivery Platform in the carrier network enables service providers to rapidly turn-up high value, managed network-based services at a fraction of the capital and operational costs of CPE-based solutions. This enables service providers to gain a least-cost service delivery and support structure. Additionally, it enables them to gain higher margins and more market share than competitors utilizing traditional service delivery mechanisms — even while offering managed firewall services at a lower customer price point.

As enterprise customers gain confidence in the WAN providers' ability to

deliver managed firewall services, a more scaleable and cost-effective service delivery model must be employed. Moving the intelligence of the service off of the customer premises and into the WAN is an effective strategy to accomplish this. Managed, network-based firewall services provide the same feature/functionality of a CPE-based service while greatly reducing capital and operational costs, as well as complexity.

The managed, network-based firewall service model enables WAN service providers to minimize service creation and delivery costs. This model virtually eliminates the need for onsite installation, configuration, and troubleshooting truck rolls, drastically reducing operational costs. This lower cost structure creates opportunities to increase revenues and/or gain market share by value-pricing the service. Services can be rapidly provisioned via a centralized services management system, shortening delivery cycles and enabling service providers to begin billing immediately. Additional services can be rapidly crafted and deployed via the same efficient delivery mechanism.

The network-based service model is a rapid and cost-effective way for service providers to deploy high-value managed firewall solutions. This model requires a comprehensive service delivery platform consisting of robust network hardware, an intelligent and scaleable services management system, and a feature-rich Customer Network Management (CNM) tool to mitigate customers' fears of losing control of network security.

In the above discussion and in the attached appendices, the term "computer" is defined to include any digital or analog data processing unit. Examples include any personal computer, workstation, set top box, mainframe, server, supercomputer, laptop or personal digital assistant capable of embodying the inventions described herein.

Examples of articles comprising computer readable media are floppy disks, hard drives, CD-ROM or DVD media or any other read-write or read-only memory device.

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiment shown. This application is intended to cover any adaptations or variations of the present

[illegible]